Automotive ECU Development

An insider's look

22 October 2021

Eugenio Grima

Greetings

- Who am I ?
- Why am I here ?
- My background
- My job

Outline

Automotive
- What is it ?
- Why so much hype?

ECU
- Development Process
- Engineering and ECU

Safety
- ISO 26262
- ASIL Assessment Process

ECU Software
- General Architecture
- Messages
- Development Process
- MISRA
- RTOS

Famous Fails
- Toyota unexpected acceleration
- Jeep remote hacking

Cybersecurity
- Basic concepts

Disclaimer

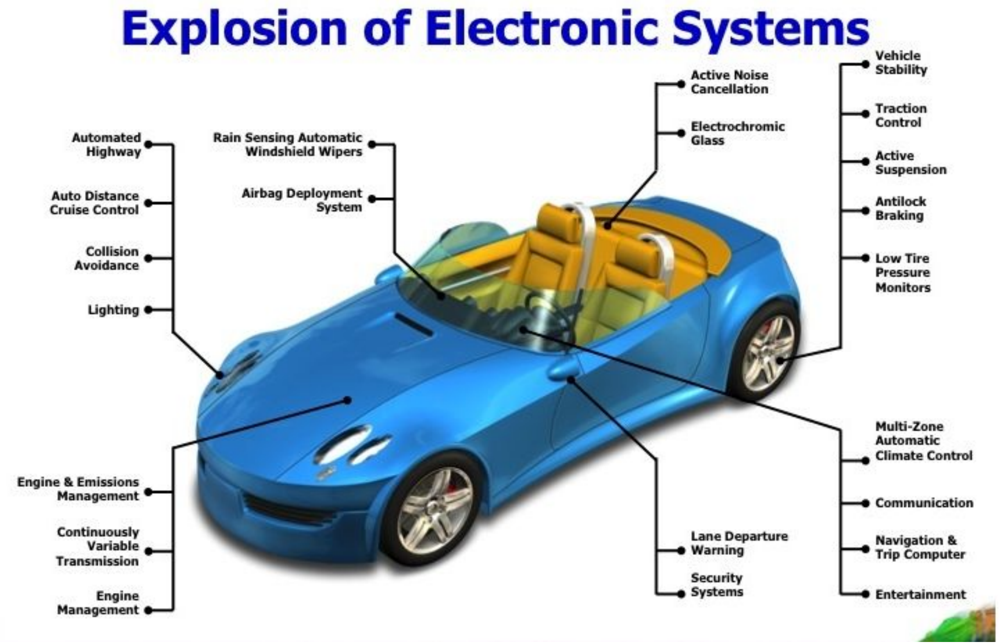
Due to **NDAs** I've signed with companies and car maker I've worked with, all informations, block diagram and technical content present in this presentation have been either found on the web or already published into magazines or produced ad-hoc as academic example.

I cannot provide any details about technical solution adopted in commercial design, nor "talk too much" about projects I've followed. **I'm sorry**.

Moreover, I don't want to make any direct or indirect advertising to software/hardware supplier, so I'll try not to mention any of them **voluntarily**.

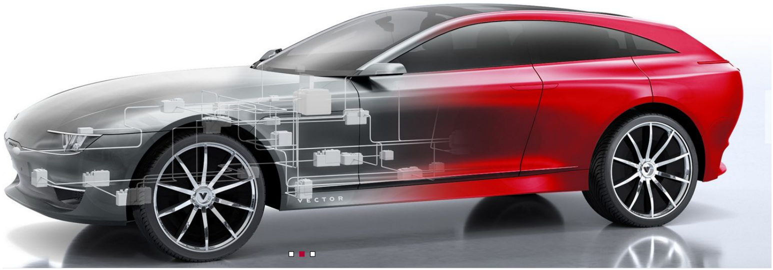
Automotive

- Electronic engineer's paradise
- Lots of technology
- Wide area of study
- Entrustable safety devices
- Focus only on electronics and software



Automotive

Today's cars have more than 20 embedded systems connected each other.



ECU: Electronic control unit. Embedded system that has input and outputs and achieve **some specific task** (e.g. Engine Control, Infotainment, Safety functions, Telemetry, Confort, etc.)

Automotive quality requirements are **closer** to **military grade** products than consumer.

Car are supposed last at least 10 years, so the **design** and **manufacturing** shall be **good enough** to guarantee such durability (automotive grade design and component!)

ECU (Electronic Control Unit)

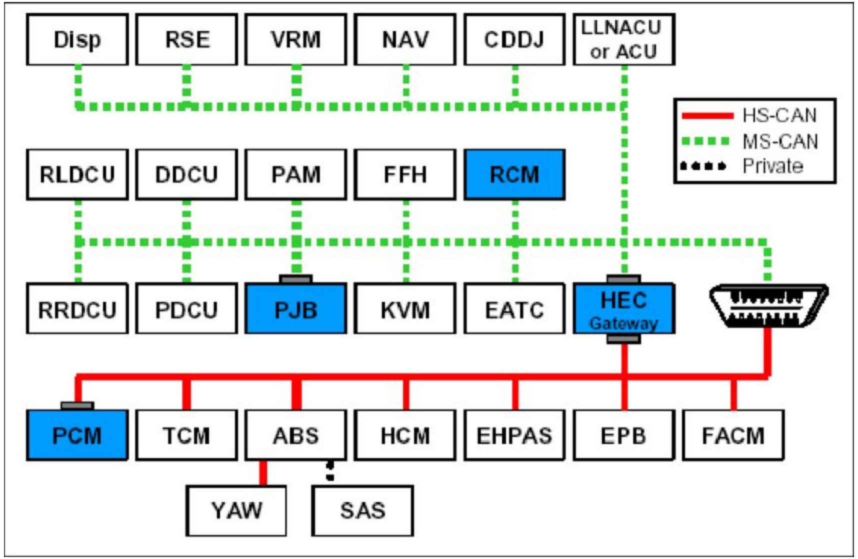
ECUs are **target oriented** embedded device that performs one (or few) specific task.

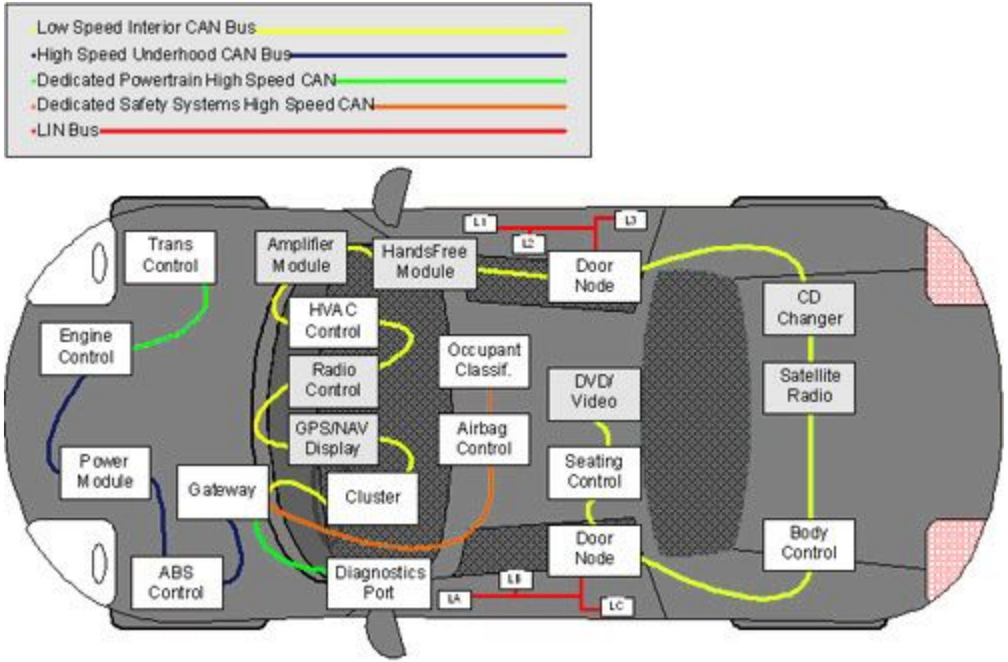
ECUs are connected using automotive bus:

- ➔ **LIN** : Single wire, Master-slave, low speed, low cost, non fault-tolerant
- ➔ **CAN** : Double wire, prioritized messages, medium/high speed, medium cost, fault tolerant
- ➔ **CAN-FD** : Enhanced CAN, non fixed sized, higher speed, support for encryption
- ➔ **Flexray** : Multimedia oriented, higher speed
- ➔ **Ethernet** : Next generation

ECU

ECUs connection can have a rather **complex** network topology, with **gateways** interconnecting more networks.





ECU : Development Process

Mechanical decisions:

- ➔ Size
- ➔ Form factor
- ➔ External interfaces
- ➔ In-vehicle arrangement

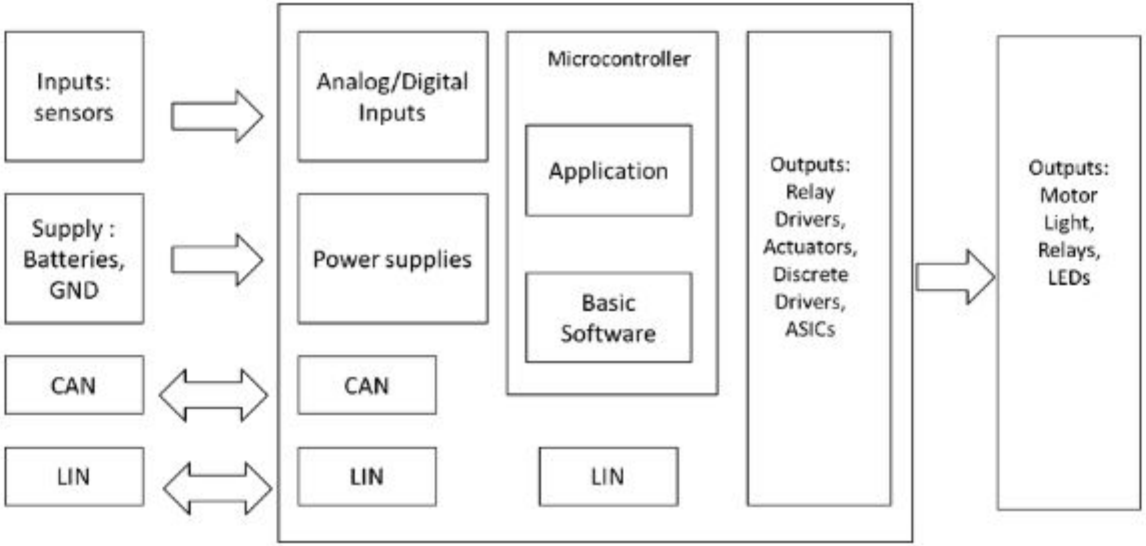
Split of jobs:

- ➔ **System** engineering: defines the system requirements
- ➔ **Hardware**: designs hardware, based on system requirements
- ➔ **Software** engineering: defines software requirements, based on system requirements
- ➔ **Software** validation: defines the validation strategy, based on system requirements
- ➔ **Safety** system: defines safety requirements, based on system requirements
- ➔ **System validation**: defines electrical validation strategy

ECU : Inner view

Hereby you can see the block diagram of a **very generic ECU**.

According to its specific task, each block is implemented, but the general structure is like this.



V-Model

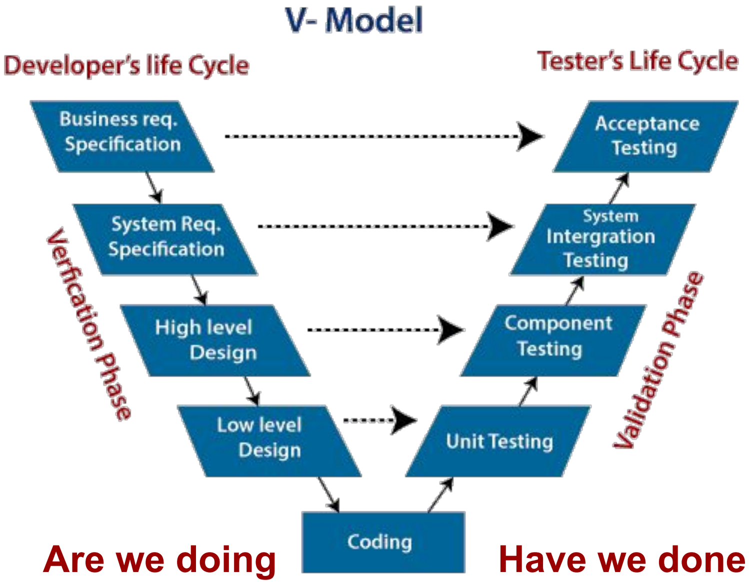
V-Model is the **software** oriented **development process** used in automotive, given by the Automotive SPICE.

The **left** hand side is **designers' job**.

The **right** hand side is **testers' job**.

In chain, the following documents shall be produced:

- ➔ SYRS, SRS, SHLD, SLLD
- ➔ SLLTP, SHLTP, SITP, SYTP



Are we doing the right things?

Have we done the things rightly?

Requirements

Requirement is a singular documented physical or functional need that a particular design, product or process aims to satisfy. (Wikipedia)

Atomic: One and only one complete feature per requirement

Verificable: The requirement can be verified

Example:

- LED1 shall blink (NOT OK: generic, not complete)
- LED1 shall be small (NOT OK: not verificable)
- If the user presses BTN1, LED1 shall blink at a frequency of 1Hz with PWM of 67%, else LED1 shall stay OFF (NOT OK: more than one feature)
- If the user presses BTN1, LED1 shall blink at a frequency of 1Hz with PWM of 67% (OK!)

Safety related requirements

The **Safety** Engineer (or Safety Manager) is in charge to analyze all the requirements and, if needed, **add "safety"** requirements to the design.

Such requirements might be added at **any level** of the design.

Example:

- **System**: The airbag shall not explode if a crash occurs at less than 30 km/h
- **Hardware**: The ECU shall contain two or more watchdogs
- **Network**: CAN message #123 shall contain redundant and opposite signals about sensor X
- **Software**: State variable of FSM123 shall be represented with values with Hamming distance greater than 3.

ISO 26262

ISO 26262 is an international standard for functional safety of electrical and/or electronic systems that are installed in serial production road vehicles. It is composed by 12 parts and conceptually similar to DO-178 used in avionics.

Item
: Specific system (or combination of systems) to which the ISO 26262 Safety Life Cycle is applied, that implements a function (or part of a function) at the vehicle level.

Element
: Either a system, a *component* (consisting of hardware parts and/or software units), a single hardware part or a single software unit — effectively, anything in a system that can be distinctly identified and manipulated.

Fault
: Abnormal condition that can cause an *element* or an *item* to fail.

Error
: Discrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition.

Failure
: Termination of an intended behaviour of an *element* or an *item* due to a *fault* manifestation.

Fault Tolerance
: Ability to deliver a specified functionality in the presence of one or more specified *faults*.

Malfunctioning Behaviour
: *Failure* or unintended behaviour of an *item* with respect to its design intent.

Hazard
: Potential source of *harm* (physical injury or health damage) caused by malfunctioning behaviour of the *item*.

Functional Safety
: Absence of unreasonable risk due to *hazards* caused by malfunctioning behaviour of Electrical/Electronic systems.

wikipedia.org

ASIL Level Assessment Process

It is a process aimed to give a “**score**” (classify) to hazards:

- ➔ **ASIL A**: Lowest hazardous (few to none injuries).
- ➔ **ASIL D**: Highest hazard (serious injuries to death for driver, passenger, external people)

The ASIL classification is the result of combination of different factors:

Severity (S):

- **S0** No Injuries
- **S1** Light to moderate injuries
- **S2** Severe to life-threatening (survival probable) injuries
- **S3** Life-threatening (survival uncertain) to fatal injuries

Exposure Classifications (E):

- **E0** Incredibly unlikely
- **E1** Very low probability (injury could happen only in rare operating conditions)
- **E2** Low probability
- **E3** Medium probability
- **E4** High probability (injury could happen under most operating conditions)

Controllability Classifications (C):

- **C0** Controllable in general
- **C1** Simply controllable
- **C2** Normally controllable (most drivers could act to prevent injury)
- **C3** Difficult to control or uncontrollable

ASIL Level Assessment Process

Severity (S):

S0 No Injuries

S1 Light to moderate injuries

S2 Severe to life-threatening (survival probable) injuries

S3 Life-threatening (survival uncertain) to fatal injuries

Exposure Classifications (E):

E0 Incredibly unlikely

E1 Very low probability (injury could happen only in rare operating conditions)

E2 Low probability

E3 Medium probability

E4 High probability (injury could happen under most operating conditions)

Controllability Classifications (C):

C0 Controllable in general

C1 Simply controllable

C2 Normally controllable (most drivers could act to prevent injury)

C3 Difficult to control or uncontrollable

		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

QM: Quality Managed

ECU : General Software Architecture

In ECUs, it is very common that **one** or **more microcontrollers/CPUs** are present.

Automotive uC are slightly different from consumer models:

- **Certified silicon** for wider temperature range
- **Deeper** production tests
- **Cybersecurity** features (often given under NDA)

uC suppliers often provide (or **suggest**) certified low level driver and certified development toolchain.

Certification is often the keyword for uC/toolchain/drivers adoption.

ECU : General Software Architecture

Software architecture is very important into automotive (and in general) software development.

Good architecture, modularly designed, must be used as much as possible.

It helps into:

- ➔ **Requirement traceability** (often more important that good unit implementation!)
- ➔ **Software recycling** (often more important that good unit implementation!)
- ➔ **Software porting** to different architectures
- ➔ Good **Powerpoint slides** (always more important that good unit implementation!)

ECU : General Software Architecture

Each **Tier 1** is somehow free to choose its own software architecture, unless strictly required to use **Autosar** architecture.

Autosar is an "**open** and **standardized** software architecture for automotive ECU" (Wikipedia).

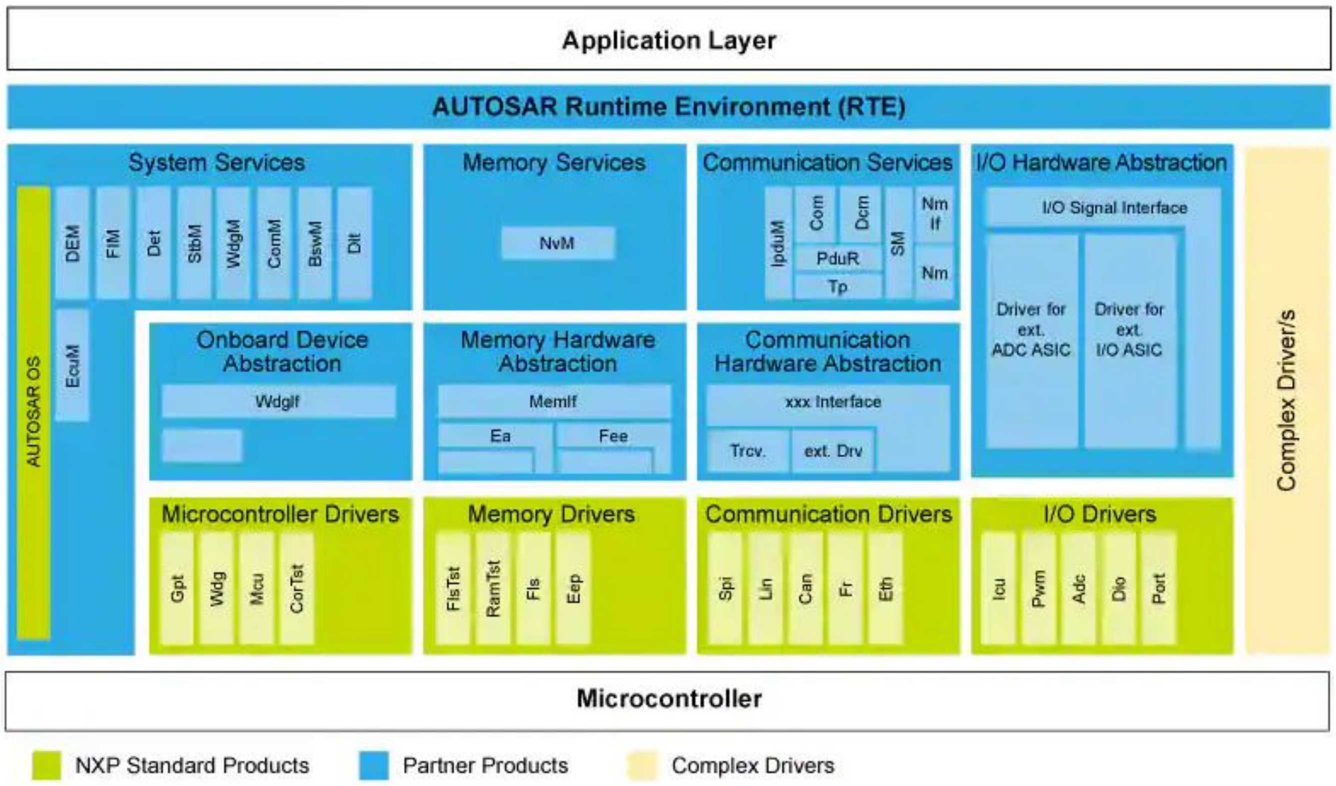
It is a layered and modular software architecture that any developer - with the right skill - can implement.

But in **real cases**, it is convenient (and sometimes imposed by car maker) to **buy** a "ready to use" **Autosar** stack:

- Developed by authoritative software house
- Documentation and requirement tracking already performed by third part
- "Free" debug (free in terms of development time)

The price for an Autosar stack can be couple of **hundreds of k€** (150-350 k€)

Autosar Architecture



In Vehicle Messaging

We've seen that there are several buses type and networks in a vehicle.

In-vehicle **messages** are always **predetermined** messages, in terms of size and content.

Messages should me as much **deterministic** as possible. **No** usage of **dynamic** objects!

Each message has its own **unique ID**, whose number is not random: in CAN **lower ID** has **higher priority** (e.g. airbag messages are more important than CD player).

Messages are internally split into **signals**, which are groups of bits with a semantic meaning.

Messages are described with standard format:

- ➔ LDF (Lin Descriptor File)
- ➔ DBC (DataBase CAN)

In Vehicle Messaging

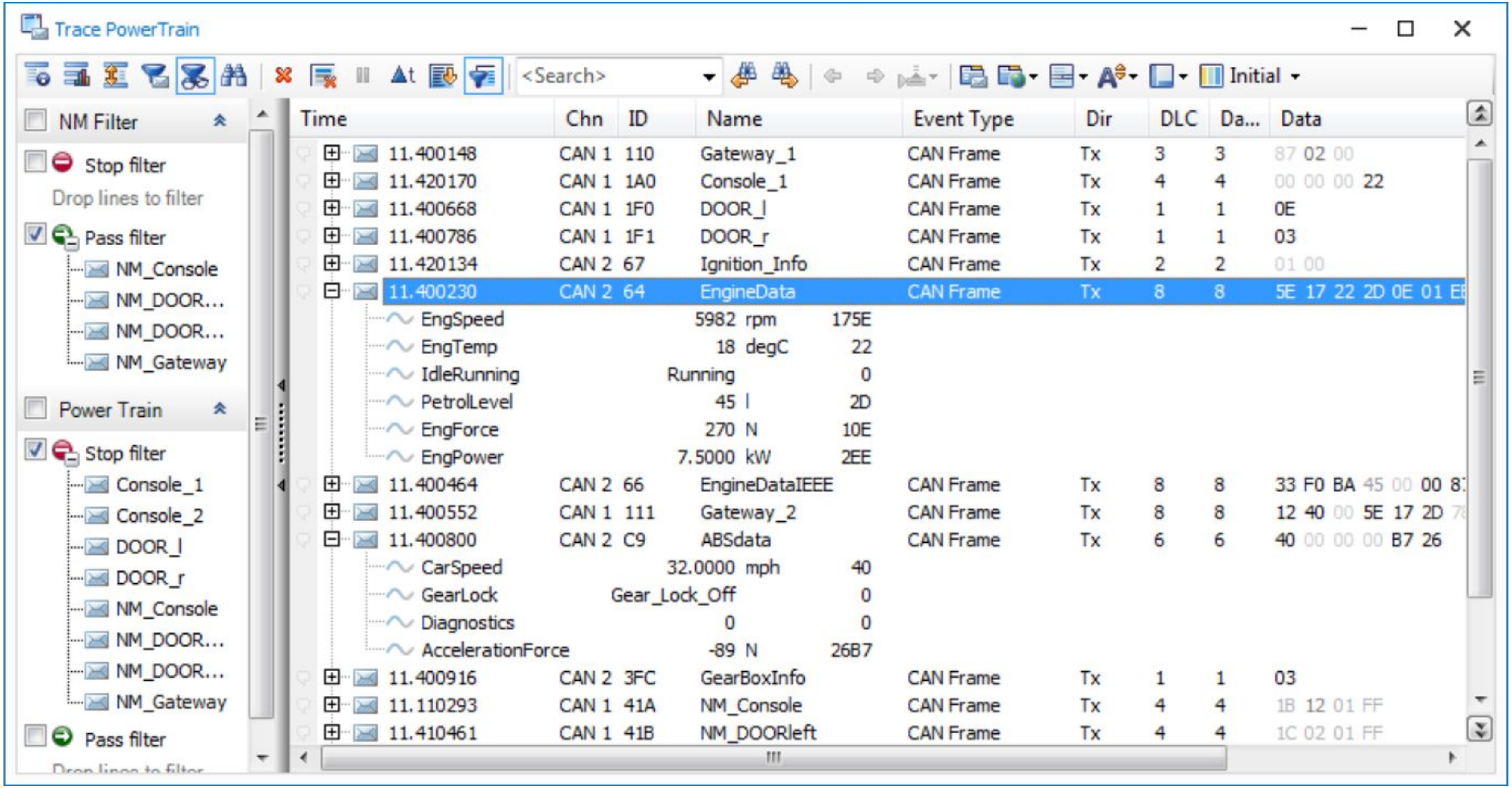
Messages running in vehicle can be divided into three **semantic categories**:

➔ **Application** messages:
- Messages that are used to archive the “usage” of the car.
- ECUs communicate each other to notify about signals value (e.g. sensors values, driving events, etc.).
- Usually, these kind of message are periodic (10ms - 5 seconds) and are used to replace “old fashion cable”

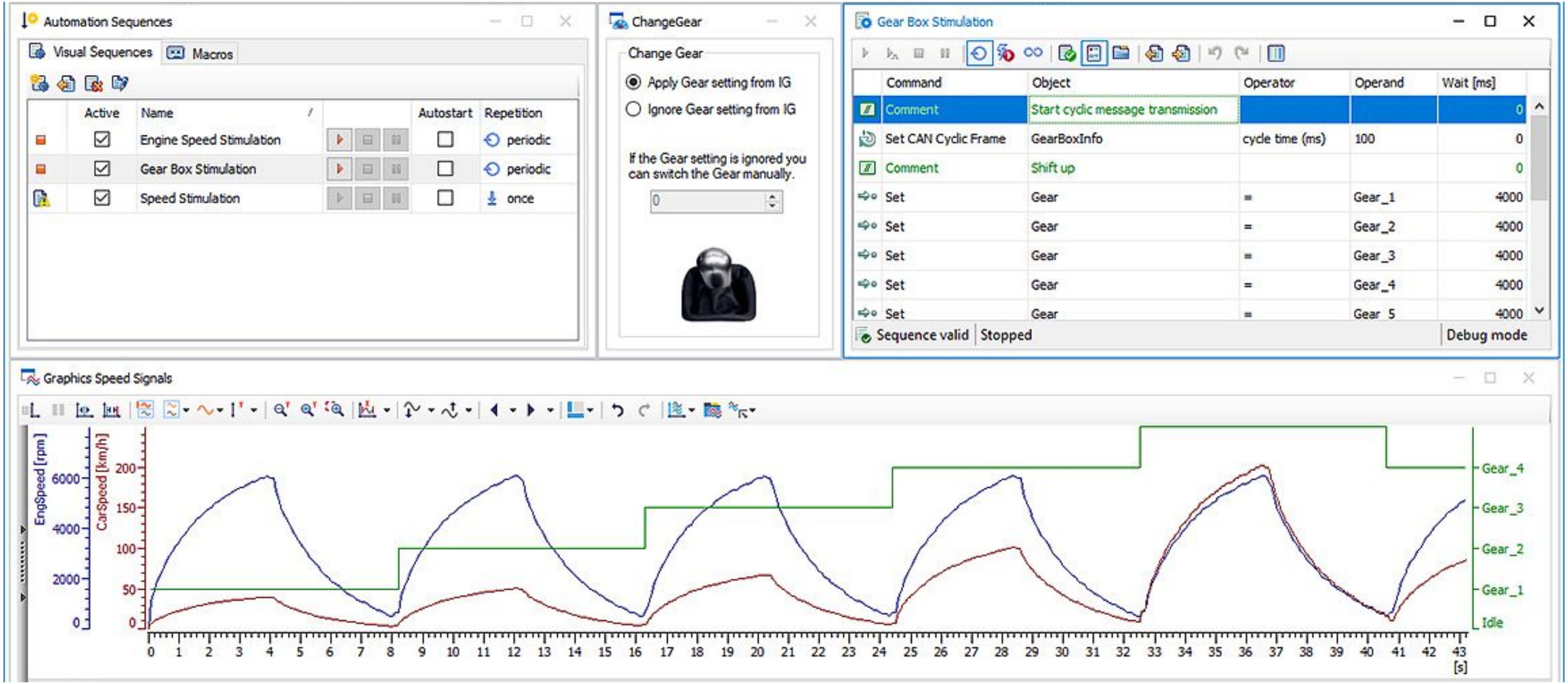
➔ **Network** messages:
- Messages that are used to drive the power status of the network(s) and the related ECUs.
- ECUs shall consume as less electric power as possible, specially when the car has the engine turned off.
- ECUs are supposed to drain <100uA in sleep (engine off).
- Network messages are designed to drive the ECUs in different power state.

➔ **Diagnostic** messages:
- Diagnostic messages are (should be) only present at service.
- Today’s main protocol is UDS (ISO 14229): several services are standardized and the technician can perform diagnosis and tune the car via UDS.

In Vehicle Messaging

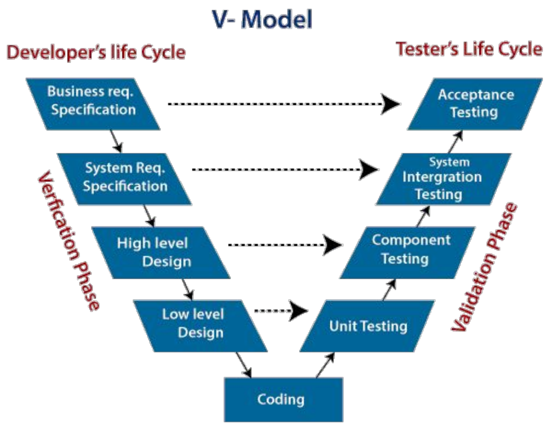


In Vehicle Messaging



Software Development

ECU software development starts with the **decomposition** of System Requirement into (several) SHLD (Software High Level Design) requirements.



SHLD describes the **architecture** of the SW, in terms of modules, hierarchy and interactions.

SHLD is exploded into SLLD (Software Low Level Design) requirements.

SLLD requirements describe the **behaviour** of the modules, by giving also details of **implementations**.

At the end - and only at the end - , coding starts :-)

Software Coding

Automotive software coding **shall** follow a (huge) **set** of **rules**: MISRA-C/C++.

Apart from automotive, MISRA rules are anyway **best practices** for programming.

MISRA-C have evolved during years with different releases.

Rules are divided between “**Advisory**”, “**Required**” and “**Mandatory**”, organized in chapters and each rule is given with its own rationale and examples.

Code compliance to MISRA is performed by (**expensively paid**) tools.

Each rule could be occasionally **violated** with the right **justification**, if strictly needed.

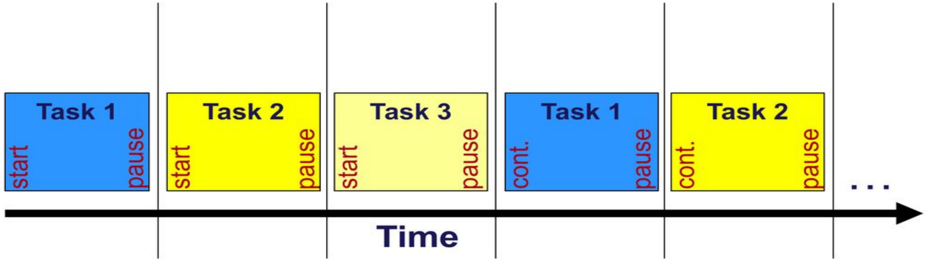
Software: RTOS

ECUs' software is often based on **RTOS**, where RT stands REALLY for **Real Time**.

Real time means time determinism (NOT fast response!).

- **Soft RT**: Missing a deadline does **not cause mission failure**, but only a temporary degradation of performances.
  E.g. in CD player, changing the track might occur - let's say - 100 ms later.
- **Hard RT**: Missing a deadline **causes mission failure**, which might also be dangerous situation.
  E.g. If the airbag explodes 100ms later than required, it is useless (or even harmful!)

Preemption is avoided if possible and task uses cooperative scheduling.



Time slots are designed according to **time-requirements**: I/O response time, messages timing.

Tasks' time is **pre-computed** at **design time** to fit the assigned time slot, by choosing the longest computation path.

If the computation time does not fit the provided slot, algorithm is split into more time slots.

Determinism is in any case a **must** for the system.

Is safety and good design overrated?

In 2009-11 Toyota Prius suffered of "**unattended acceleration**" and some tens of people died (brakes were not strong enough to stop the movement!)

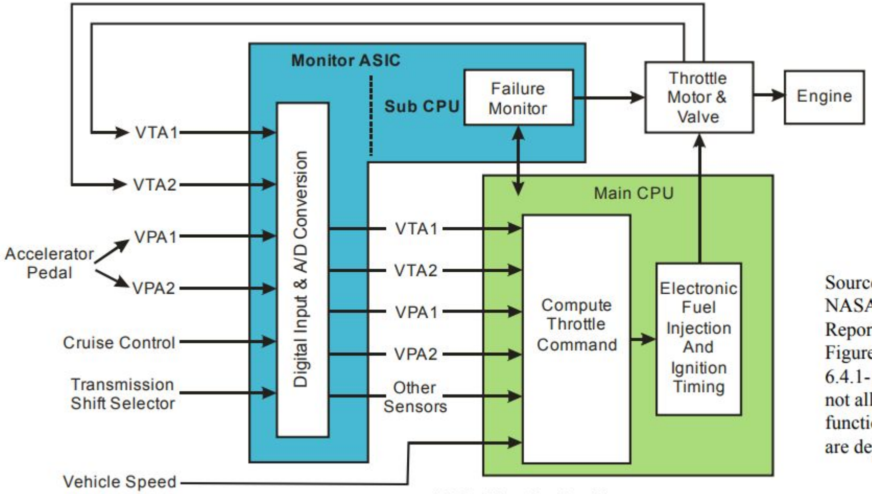
NASA has analyzed the design and issues on software and hardware have been found!

Software: several **MISRA rules violation** including usage of recursion.

Hardware: only the "Monitor ASIC" receives the Accelerator Pedal input, and, although the pedal signal is redounded on two different input, the **ADC is unique for both inputs.** The Main CPU receives the VPA signal ONLY from Monitor ASIC.

ETCS Architecture (simplified)

256.6K Non-Comment Lines C Source + 39.5K NCSL headers (Main CPU) + Proprietary Monitor Chip software [NASA App. A p. 21]



Source: NASA UA Report Figure 6.4.1-1; not all functions are depicted

VTA: Throttle Position
VPA: Accelerator Pedal Position

Our previous examples overcomes these flaws ;-)

Cybersecurity in Automotive

In 2015, a flaw in Jeep infotainment allowed remote attackers to move the steering wheel, play with cluster and even disable breaks (while driving!).

Full Video at : https://www.youtube.com/watch?v=MK0SrxBC1xs

Cybersecurity in Automotive

This episode led most of car makers to give **higher importance** to **cybersecurity**, giving some specific requirements.

Secure Boot: Bootloader shall start only signed firmware

Secure Download: Bootloader shall not allow to install unsigned firmware

Signed Diagnostic Services: Car services shall have their own software keys

Message Signing: In vehicle messages shall be signed

Encrypted Messages: In vehicle messages shall be encrypted

Thanks for your attention

Eugenio Grima
eugeniogrima@gmail.com